On the Use of Concurrent Multipath Transfer over Asymmetric Paths

Thomas Dreibholz, Martin Becke, Erwin P. Rathgeb
University of Duisburg-Essen
Institute for Experimental Mathematics
Ellernstraße 29, 45326 Essen, Germany
{dreibh,martin.becke,rathgeb}@iem.uni-due.de

Michael Tüxen
Münster University of Applied Sciences
Dept. of Electr. Engineering and Computer Science
Bismarckstraße 11, 48565 Steinfurt, Germany
tuexen@fh-muenster.de

***Abstract*—With the deployment of more and more resilience-critical Internet applications, there is a rising demand for multi-homed network sites. This leads to the desire for simultaneously utilising all available access paths to improve application data throughput. This is commonly known as Concurrent Multipath Transfer (CMT); approaches for several Transport Layer protocols have been proposed. Combined with Resource Pooling (RP), CMT can also fairly coexist with concurrent non-CMT flows. Current approaches focus on symmetric paths (i.e. similar bandwidth, delay and error rate). However, asymmetric paths are much more likely – particularly for realistic Internet setups – and efficient CMT usage on such paths is therefore crucial.**

In this paper, we first show the challenges of plain as well as RP-aware CMT data transport over asymmetric paths. After that, we introduce mechanisms for efficient transport over such paths. Finally, we analyse the performance of our approaches by using simulations.[1][2]

Keywords: Concurrent Multipath Transfer, Resource Pooling, Asymmetric Paths, Buffer Handling, Performance Analysis

I. Introduction

During the first experimental network communications tests by just connecting two nodes by a cable, nobody could have predicted how this technology was going to change the handling of devices and – in result – the world. Nowadays, IP-based network communication has emerged to the central pillar of the information society. Even mobile phones provide ubiquitous Internet access over different interfaces – via different networks and paths – by using different ISPs.

Switching between different network interfaces is a common approach to provide mobility or increased availability. However, utilising all interfaces *simultaneously* – in order to increase throughput by load sharing [1] – is still not widespread in the Internet. There is a growing consensus on the benefits of this so-called Concurrent Multipath Transfer (CMT) [2] and a considerable interest in combining multipath routing with rate control [3], [4] on the Transport Layer. Technical advances, like a TCP-friendly congestion control for multipath transport in a Resource Pooling (RP) [5], [6] manner set the stage for efforts in communities – like the Multipath TCP Working Group of the IETF – to extend existing transport protocols like TCP and SCTP [7] with CMT.

Current research on plain or RP-aware CMT performance proves the throughput benefits for using symmetric paths [1], [2], [6]. However, the challenge is an efficient transport over asymmetric paths – i.e. paths with different bandwidths, delays and error rates – which are highly realistic for multi-homed Internet sites. In this paper, we first demonstrate the limitations of state-of-the-art CMT on asymmetric paths. We furthermore introduce mechanisms for efficient CMT transport in such scenarios and show their performance by using simulations.

[1]Parts of this work have been funded by the German Research Foundation (Deutsche Forschungsgemeinschaft – DFG).

[2]The authors would like to thank Irene Rüngeler for her friendly support regarding INET and Randall Stewart for initial discussions about Receiver Buffer Splitting.

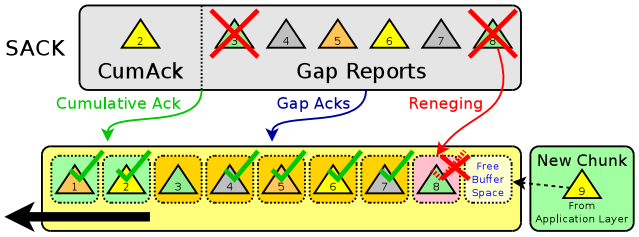

Figure 1. Retransmission Queue and Selective Acknowledgements

II. Reliable User Data Transport in Multi-Homed Environments

In this paper, we exemplarily use the SCTP protocol [7], since it is widely known and its basic multi-homing features are well-researched. However, the mechanisms and approaches presented in this paper are common to all CMT protocols. An adaptation to other protocols – particularly Multipath TCP – is a straightforward task.

SCTP is a general-purpose, connection-oriented, unicast transport protocol which provides the reliable transport of user messages within multiple independent streams and multi-homing out of the box. An SCTP connection is denoted as *association*. Unlike TCP, each SCTP endpoint can use multiple IPv4 and/or IPv6 addresses to transmit to its peer. Each peer address defines a unidirectional *path*.

User messages are segmented into units of so-called DATA chunks, which are identified by unique Transmission Sequence Numbers (TSN). For each DATA chunk, the application decides whether the chunk must be delivered to the peer's Application Layer within its stream in sequence (*ordered delivery* – similar to TCP) or may be delivered out of sequence (*unordered delivery*). Multiple smaller DATA chunks may be *bundled* into a single packet to reduce overhead by sending full packets. In this paper, we only use a single stream.

A Selective Acknowledgement (SACK) chunk [7, subsubsection 3.3.4] is transmitted by the receiver to acknowledge received DATA chunks and report gaps (i.e. missing DATA chunks given by their TSNs) to the sender. It also contains the receiver's advertised window size (*rwnd*; given in bytes). By default, a SACK is generated for every second packet (not chunk); it is sent back over the peer path of the last received packet. An example SACK chunk is shown in Figure 1. It consists of a *cumulative acknowledgement* (CumAck), which ac-

knowledges all TSNs including the given TSN (here: TSN #2). Optional *gap reports* indicate further already-received TSNs (GapAck; here: TSN #5 to TSN #7). But some more TSNs (here: TSN #3) are needed to perform a CumAck.

While CumAcks are obviously non-renegable, the receiver may revoke GapAcks (here: TSN #8). This may e.g. happen when the receiver queue gets too full to store earlier chunks (which are necessary for the next CumAck). Although reneging usually occurs rarely, the sender must always be prepared to reneg GapAck'ed TSNs. That is, GapAck'ed chunks must remain *buffered* in the sender buffer, although these chunks are not being *outstanding* (i.e. in flight) any more. To improve efficiency, the Non-Renegable SACK extension (NR-SACK [8]) allows signalising non-renegable GapAcks.

Two mechanisms perform retransmission of missing chunks: (1) Once a DATA chunk has been gap-reported as missing for 3 times, it is retransmitted immediately on the same path (Fast Retransmission [7, subsection 7.2.4]). (2) Further retransmissions (possibly on alternative paths) are triggered by a timer (Timer-Based Retransmission [7, subsubsection 6.3.3]). The timeout triggering a Timer-Based Retransmission is denoted as Retransmission Timeout (RTO) of a path. It is set for the earliest outstanding chunk. The RTO of a path is calculated [7, subsection 6.3] by the Round-Trip Time (RTT) of that path, with lower and upper thresholds RTO.Min=1s and RTO.Max=60s (default settings from [7, section 15]).

On each path, SCTP applies AIMD (Additive Increase, Multiplicative Decrease) congestion window control [6, section 3] – similar to TCP. Standard SCTP [7] uses only one path in each direction to transmit DATA chunks. This selected path is denoted as *primary path*. Alternative paths are only used for retransmissions; the primary path may be changed e.g. in case of path errors. This ensures fairness against concurrent TCP/SCTP flows which share a single bottleneck link with a multi-homed SCTP association [6].

CMT-SCTP [2] denotes the realization of CMT for SCTP. To ensure fairness against concurrent single-homed flows (e.g. TCP flows), CMT-SCTP requires all paths to be disjoint. While this is possible for well-designed intranets, this property cannot be ensured for arbitrary Internet connections. By combining CMT-SCTP with RP [5] to CMT/RP-SCTP [6], a fair CMT deployment in such cases is also possible – at the cost of a slower congestion window growth rate [6, section 4].

SCTP uses a checksum to protect packets against bit errors; damaged packets are dropped. Since this leads to a congestion window reduction, the "Packet Drop Reporting" extension [9] allows a receiver to inform its sender about damaged packets. Then, the sender can retransmit them – without changing the congestion window. Therefore, the throughput is not reduced by misinterpreting damaged packets as congestion indications. However, this extension can only be applied if packets being damaged are still delivered to the remote SCTP stack.

III. Challenges of Asymmetric Paths

A. Sender Buffer Blocking

Figure 2 illustrates a problem which we denote as *Sender Buffer Blocking*: on path #1, the TSN #27 is successfully received – as well as the TSNs #29 to #34 on path #2. This allows a CumAck of TSN #27 and GapAcks for TSNs #29 to #34. The delayed TSN #28 is missing. Due to possible reneging, the chunks #29 to #34 cannot be removed from the sender buffer. Only the space of a single chunk is freed (by CumAck of TSN #27), allowing only a single more new chunk to be transmitted into the network. In the worst case,

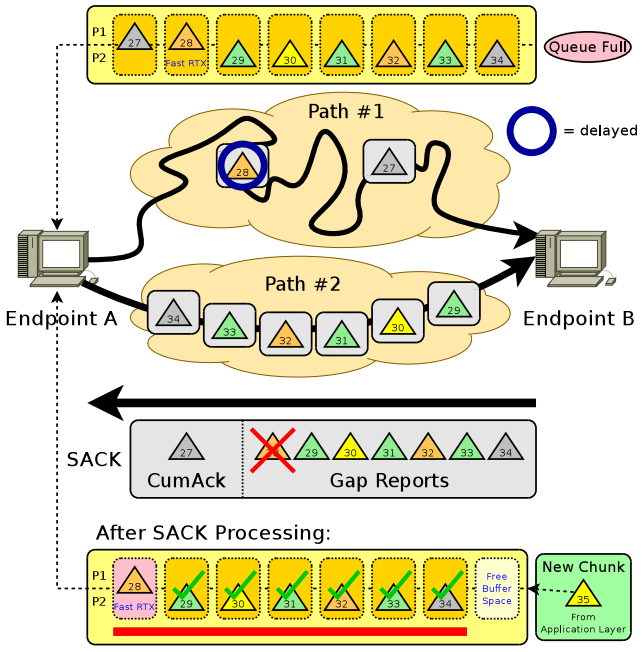

Figure 2. Sender Buffer Blocking

TSN #28 is a Fast Retransmission being lost. Then, once the buffer is fully blocked, the transmissions on *all* paths are suspended until a successful Timer-Based Retransmission of this chunk. NR-SACKs [8] are able to reduce the problem for certain cases, but cannot solve it entirely – as we will show in subsection VI-B.

B. Receiver Buffer Blocking

On the receiver side, a blocking problem – which we denote as *Receiver Buffer Blocking* – can occur. Similar to Sender Buffer Blocking, missing chunks preventing other already-received ones to be deliverable to the application result in blocking buffer space. Obviously, chunks being received in the expected sequence can leave the buffer (by providing them to the application). However, further chunks may only leave the buffer for unordered transmission. Due to in-sequence delivery, ordered chunks have to wait until all previous gaps in the sequence have been filled. In [10] it is shown that specific retransmission strategies can be used to alleviate some of the throughput reduction related to Receiver Buffer Blocking.

C. Spurious Fast Retransmission Bursts

The occasional appearance of Timer-Based Retransmissions causes a problem when paths have different delays. According to [7, subsubsection 6.3.3], *any* outstanding chunk on the affected path "should be marked for retransmission", which is – in case of multi-homing – on another path. This behaviour is useful, since it shifts away traffic from a possibly broken path to a working one. But if such an outstanding chunk with TSN c is just delayed, TSN c will exist in the network twice. If the original transmission of c is acknowledged – which appears to the sender as an acknowledgement of c on the *new* path – and if c somewhat differs from the TSN range on the new path (i.e. it is lower or higher, due to the delay difference between the paths), the CMT Fast Retransmission handling [2, subsection III-A] may assume large gaps in the TSN sequence on the new path. These putative gaps trigger bursts of Fast Retransmissions. While the resulting duplicate chunks are

simply dropped by the receiver, the Fast Retransmissions lead to congestion window reduction.

IV. Efficient Transport over Asymmetric Paths

A. Buffer Size Considerations

The buffer sizes of sender and receiver are important system parameters. The AIMD congestion control behaviour [6] leads to frequent losses of single packets (which is intended behaviour!). These losses are handled by Fast Retransmissions, which should therefore be covered by the buffer to avoid significant interruptions. That is, the minimum buffer size $B_{\min}$ – for the sender as well as for the receiver buffer – in a setup with paths $P = \{P_1, \ldots, P_n\}$, BW_i the bandwidth and RTT_i the RTT of path P_i is:

$$B_{\min} = 2 * \max_{1 \le i \le n} (\mathrm{RTT}_i) * \sum_{i=1}^{n} \mathrm{BW}_i.$$

Timer-Based Retransmissions occur in case of high congestion (i.e. they should be rare). To cover a Timer-Based Retransmission, the minimum buffer size $B_{\min}$ for the sender and receiver buffers is:

$$B_{\min} = (3 * \max_{1 \le i \le n} (\mathrm{RTT}_i) + \max_{1 \le i \le n} (\mathrm{RTO}_i)) * \sum_{i=1}^{n} \mathrm{BW}_i.$$

That is, in the worst case it takes 3 times the RTT (first transmission, Fast Retransmission, Timer-Based Retransmission) plus the highest path RTO. Since the default *minimum* RTO is RTO.Min=1s [7, section 15], the Timer-Based Retransmission coverage by the buffer space is usually too expensive (e.g. about 32 MiB – per association – in a 100/100 Mbit/s setup with RTO=1s and RTT=110ms).

B. Buffer Splitting

To avoid one path occupying too much buffer space – which prevents other paths from sending out new chunks – our approach denoted as *Sender Buffer Splitting* simply splits the sender buffer of size B^{Sender} into n (i.e. number of paths) sections. Let $\mathrm{Buffered}_i$ be the buffer size occupied by chunks on path P_i and MTU_i be the MTU on path P_i. Then, a new chunk on path P_i may be sent if its buffer share allows another MTU-sized packet:

$$\mathrm{Buffered}_i + \mathrm{MTU}_i \le \frac{B^{\mathrm{Sender}}}{n}. \quad (1)$$

Similar to the sender buffer handling, the sender is also able to care for the receiver buffer (by taking notice of the advertised window size Rwnd). Let $\mathrm{Outstanding}_j$ be the buffer size occupied by outstanding (i.e. still unacknowledged) chunks on path P_j. Using *Receiver Buffer Splitting*, a new chunk on path P_i may be sent if:

$$\mathrm{Buffered}_i + MTU_i \le \frac{\mathrm{Rwnd} + \sum_{j=1}^{n} \mathrm{Outstanding}_j}{n}. \quad (2)$$

C. Chunk Rescheduling

Our approach denoted as *Chunk Rescheduling* copes with the problem of some delayed or lost chunks stalling the whole transmission (see subsection III-A). On each transmission event on a path P_i, the so-called *blocking fraction* of path P_i is calculated as:

$$\mathrm{BlockingFraction}_i = \frac{\mathrm{Buffered}_i - \mathrm{Outstanding}_i}{B^{\mathrm{Sender}}}$$

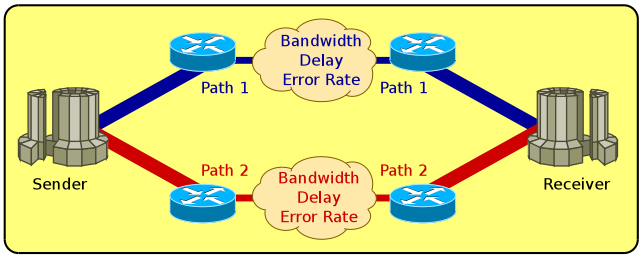

Figure 3. The Simulation Setup

Chunk Reschedulung is triggered when P_i blocks more than half of the path's buffer share, i.e.:

$$\mathrm{BlockingFraction}_i > \tfrac{1}{2} \quad \text{(No Buffer Splitting)}$$
$$\mathrm{BlockingFraction}_i > \tfrac{1}{2} * \tfrac{1}{n} \quad \text{(With Buffer Splitting)}$$

Then, we look for the *first* chunk which blocks the removal of chunks on path P_i from the sender buffer. This chunk is rescheduled for retransmission on path P_i as soon as possible, i.e. when the congestion window of P_i allows its transmission. Since the initial transmission of this chunk may still be in flight on the previous path P_{p}, the congestion window of P_{p} is reduced by the chunk size. Furthermore, since it is unknown to the sender whether the chunk has been finally acknowledged on P_i or P_{p} (i.e. first transmission delayed but not lost), acknowledgements for rescheduled chunks are not considered for congestion window advances (see [2, subsection III-B] for details on CMT congestion window handling). That is, Chunk Rescheduling does not introduce any unfairness to concurrent flows. In the context of [10], Chunk Rescheduling can be seen as a preventive retransmission policy.

D. Smart Fast Retransmission

To cope with the "Spurious Fast Retransmission Bursts" problem described in subsection III-C, our approach "Smart Fast Retransmission" simply does not consider chunks being moved from another path in the decision about Fast Retransmissions on the new path.

V. Simulation Model and Setup

For our performance evaluation, we have used the OMNeT++-based INET framework with the CMT-SCTP model [11] including support for CMT/RP-SCTP [6]. The SimProcTC [12] tool-chain has been used for parametrization and results processing. The results plots in this paper show the average values of 64 runs and their 95% confidence intervals. Figure 3 illustrates the simulation setup: sender and receiver have been connected via two paths. The following configuration parameters have been used, unless otherwise specified:

- The QoS parameters (bandwidth, delay, bit error rate) of the links between the routers of each path have been configurable (default: bandwidth of 100 Mbit/s and delay of 1ms without errors). The bottleneck network interfaces use RED queues (with settings based on [13]: $w_q = 0.002$, $\min_{\mathrm{th}} = 20$, $\max_{\mathrm{th}} = 80$, $\max_p = 0.02$).
- After association establishment and transmission start, the actual throughput measurement has been started after 19s, the measurement duration has been 60s.
- The senders have been saturated (i.e. they have tried to transmit as much data as possible); the message size has been 1,452 bytes at an MTU of 1,500 bytes (i.e. MTU-sized packets [7, subsubsection 3.3.1]). Smart Fast Retransmission has been turned on.

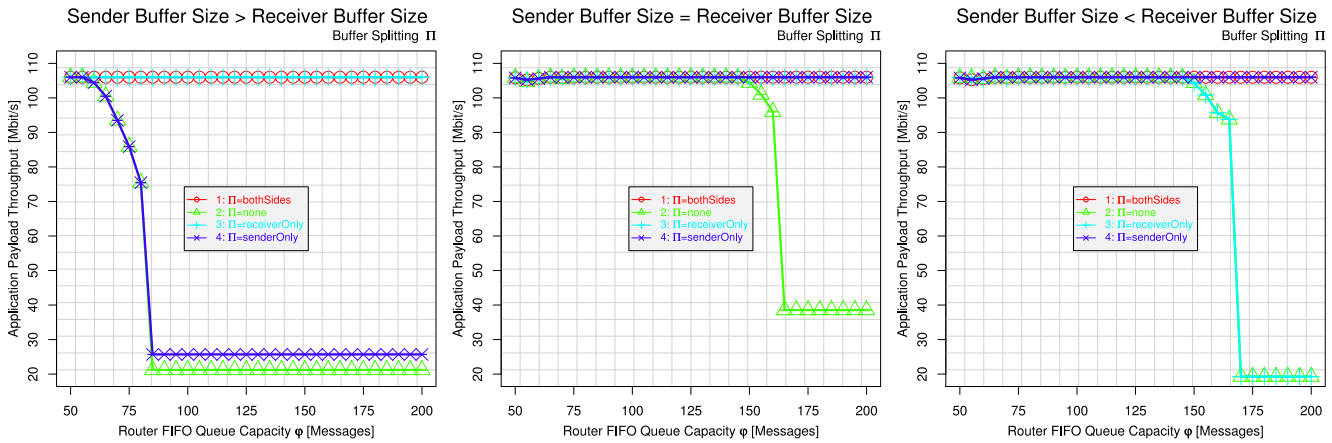

Figure 4. Buffer Splitting for Different Sender and Receiver Queue Sizes

VI. Performance Analysis

A. Avoiding the Buffer Blocking Problem

In order to illustrate the behaviour of Buffer Splitting (see subsection IV-B), we have applied FIFO router queues (since RED is non-deterministic) in the simulation setup, using 100 Mbit/s on path #1 but only 10 Mbit/s on path #2 for unordered CMT-SCTP transmission with NR-SACK enabled. The maximum router queue message capacity φ has been varied. Figure 4 presents the application payload throughput results for Buffer Splitting turned off (Π=none) as well as Sender Buffer Splitting (Π=senderOnly; see formula 1), Receiver Buffer Splitting (Π=receiverOnly; see formula 2) and both simultaneously (Π=bothSides).

If the sender buffer is larger than the receiver buffer (250,000 bytes vs. 125,000 bytes; shown on the left-hand plot), the throughput significantly sinks from about 106 Mbit/s to only 20 Mbit/s with no or just Sender Buffer Splitting applied: the growing number of outstanding messages on the slow path fills the receiver queue, while it cannot provide sufficient space for messages on the fast path. Receiver Buffer Splitting solves the problem here. Having a sender buffer smaller than the receiver buffer (125,000 bytes vs. 250,000 bytes; shown on the right-hand plot), the scenario turns. Now, the sender buffer gets blocked by the outstanding messages on the slow path – and Sender Buffer Splitting solves the problem. For equal buffer sizes (250,000 bytes for both; shown on the middle plot), the problem is solved by Sender or Receiver Buffer Splitting. That is, applying Sender *and* Receiver Buffer Splitting simultaneously (i.e. Π=bothSides) solves the Buffer Blocking in all cases. Therefore, we only apply this variant in the following and commonly denote it as *Buffer Splitting*.

B. Unordered Transmission

Figure 5 shows the application payload throughput results for unordered CMT-SCTP transmission when using a sender buffer of $2.5{*}10^5$ bytes and a receiver buffer of $1.25{*}10^5$ bytes (i.e. 2 orders of magnitude smaller than the bytes per second transported in a 100/100 Mbit/s setup!). The left-hand plot shows the results for varying the path #2 bandwidth ρ. Obviously, the throughput of regular CMT using neither Buffer Splitting (i.e. Π=none) nor NR-SACK (i.e. ν=false) falls to almost zero for highly asymmetric links (e.g. ρ=1 Mbit/s). NR-SACK alone (ν=true) just improves the situation somewhat, while Buffer Splitting alone (i.e. Π=bothSides, ν=false) makes the problem even worse. But combining both mechanisms, the throughput scales linearly and achieves the expected values (e.g. 106 Mbit/s in a 100/10 Mbit/s setup and around 192 Mbit/s in the symmetric 100/100 Mbit/s case).

Varying the path #2 bit error rate ϵ (middle plot in Figure 5) for the same buffer size settings as before, there is almost a throughput of zero for $\epsilon = 5 * 10^{-6}$. A significant performance improvement is achieved by turning on NR-SACK (i.e. ν=true). The performance is independent of Buffer Splitting usage. By using Packet Drop Reporting (i.e. Δ=true), the performance is increased further. In this case, the scenario looks similar to a symmetric setup: dropped packets are just retransmitted – without reducing the congestion window – and only the overhead of these retransmissions slightly reduces the throughput. That is, the best performance is achieved for using NR-SACKs and Packet Drop Reports, if applicable; Buffer Splitting does not change the performance in loss situations.

The AIMD congestion control behaviour of halving the congestion window on a packet loss and increasing it again by one MTU *per RTT* in congestion avoidance phase (i.e. slow growth for large RTT) leads to a reduced throughput. The existence of delay requires larger buffers in order to have a sufficient number of chunks in flight. Varying the path #2 delay δ when using a sender buffer of $2.5 * 10^6$ bytes and a receiver buffer of $1.25{*}10^6$ bytes (right-hand plot in Figure 5), the best performance is achieved for using NR-SACK (i.e. ν=true) in combination with Smart Fast Retransmission (i.e. τ=true; see subsection IV-D) – regardless of the Buffer Splitting configuration. Turning the latter feature off (i.e. τ=false), the occurrence of Spurious Fast Retransmission Bursts (see subsection III-C) leads to a significantly reduced throughput for a higher delay δ.

The application payload throughput results for unordered CMT/RP-SCTP transport – using the same scenario setups as the corresponding CMT-SCTP simulations above – are presented in Figure 6. At first, the performance is quite corresponding to the plain CMT results. However, an interesting observation is the performance difference for almost symmetric scenarios: while the CMT/RP-SCTP throughput for a path #2 bandwidth of ρ=100 Mbit/s (see left-hand plot in Figure 6) achieves only about 175 Mbit/s when Buffer Splitting is turned off (i.e. Π=none), plain CMT-SCTP (see the corresponding plot in Figure 5) reaches the expected bandwidth of 192 Mbit/s in the same situation. The reason for this deviation of the CMT/RP performance is the congestion window handling of CMT/RP-SCTP (see [6, section IV] for details), which may lead to a very slow growth rate in case of

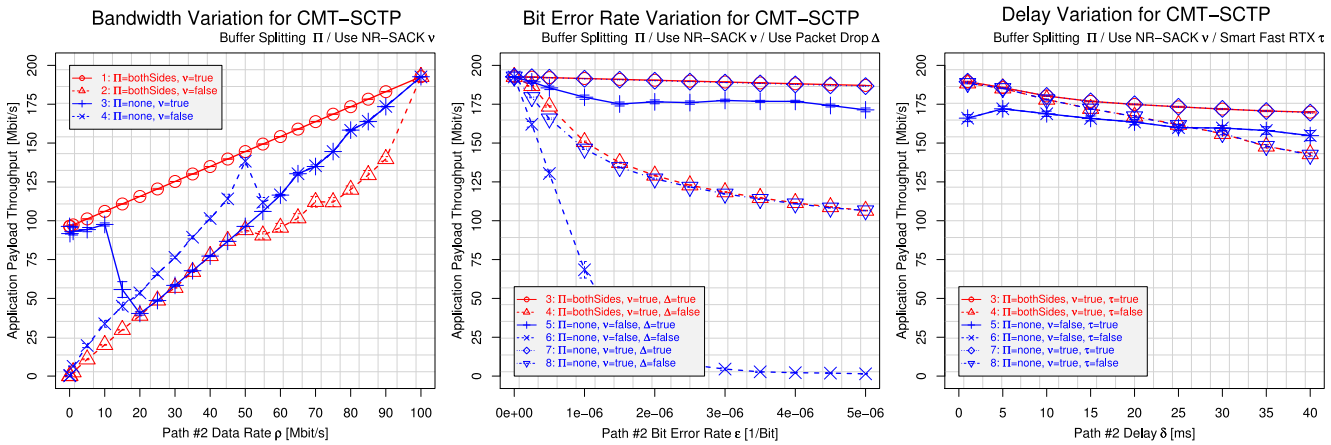

Figure 5. Unordered Transmission for CMT-SCTP

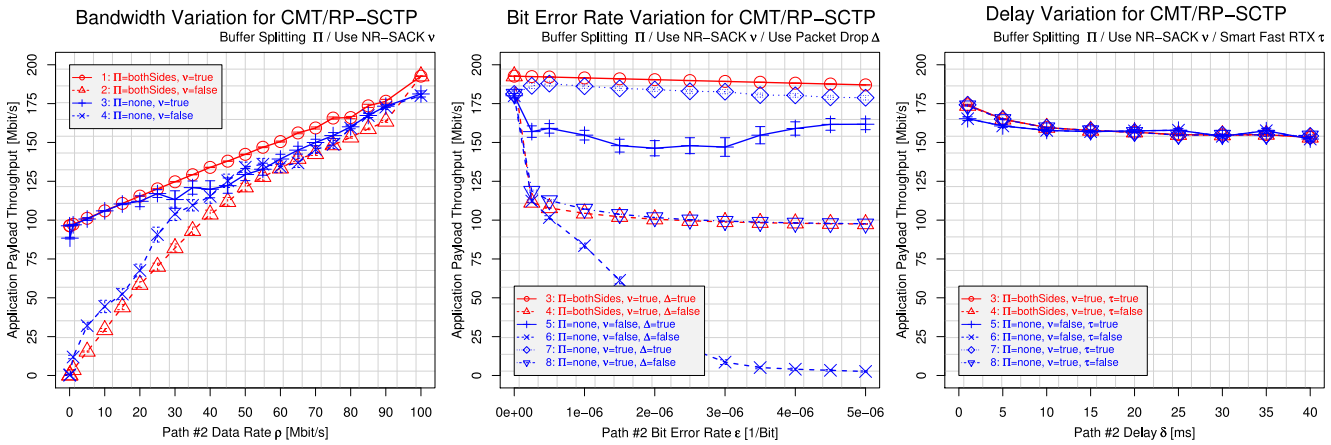

Figure 6. Unordered Transmission for CMT/RP-SCTP

temporary path capacity ratio changes. The resulting effect is similar to temporary bandwidth changes. The usage of Buffer Splitting (i.e. Π=bothSides) solves this problem of bandwidth differences, i.e. also CMT/RP-SCTP achieves the expected throughput of 192 Mbit/s. The same effect can also be observed for the path #2 bit error rate ϵ variation (middle plots): the highest bandwidth is only reached when Buffer Splitting is turned on (i.e. Π=bothSides).

In summary, effective unordered CMT as well as CMT/RP transport over asymmetric paths requires NR-SACK and Buffer Splitting. Smart Fast Retransmission improves the performance and in case Packet Drop Reporting is applicable, it should be used.

C. Ordered Transmission

The key difference between unordered and ordered transport is the fact that an user message can only be provided to the application when *all* previous messages have been forwarded before. This section considers only the completely ordered case, which is the hardest one and also applies to Multipath TCP. Simply NR-SACK'ing already received chunks cannot work, since this only mitigates Sender Buffer Blocking, but not Receiver Buffer Blocking. This leads to the need for larger buffers. For the following simulations, we have used sender and receiver buffer sizes of $2.5 * 10^6$ bytes (i.e. still one order of magnitude smaller than the $25 * 10^6$ bytes per second transferred in a 100/100 Mbit/s setup!).

Figure 7 presents the CMT-SCTP application payload throughput using ordered transport for varying the path #2 bandwidth ρ (left-hand side), bit error rate ϵ (middle plot) and delay δ (right-hand plot); the corresponding results for CMT/RP-SCTP are shown in Figure 8. The most important observation is that Chunk Rescheduling (see subsection IV-C) for ordered transmission takes over the role of NR-SACK for unordered transmission (compare the corresponding results of unordered transport in figures 5 and 6): Chunk Rescheduling (Ψ=threshold) ensures that missing or highly late chunks are quickly resent, facilitating a fast CumAck and avoiding Buffer Blocking. Combining Chunk Rescheduling with Buffer Splitting (i.e. Π=true) avoids performance decay in highly asymmetric bandwidth scenarios (e.g. $\rho \leq 1$ Mbit/s – see the left-hand plot); Packet Drop Reporting (i.e. Δ=true) ensures a high throughput in loss situations (middle plot) and using Smart Fast Retransmission (i.e. τ=true) improves the performance for different path delays (right-hand plot).

As expected, the four mechanisms also improve the performance of ordered transport using CMT/RP-SCTP (compare the corresponding results in Figure 8). In summary, effective ordered CMT as well as CMT/RP transport over asymmetric paths requires Chunk Rescheduling, Buffer Splitting and Smart Fast Retransmission. In case Packet Drop Reporting is applicable, it should be used. Our CMT extensions have been contributed as Internet Draft [14] into the IETF standardization process of SCTP.

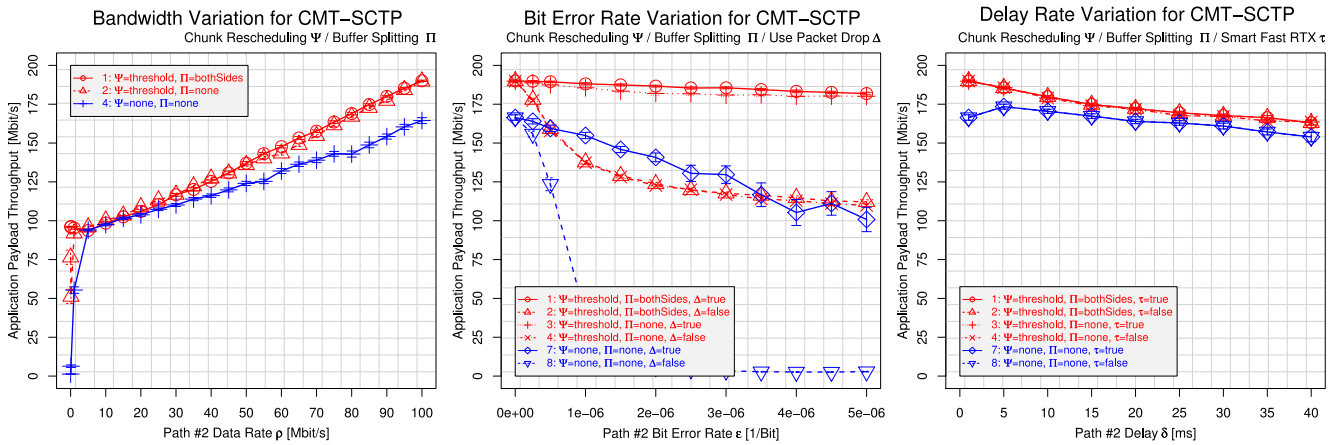

Figure 7. Ordered Transmission for CMT-SCTP

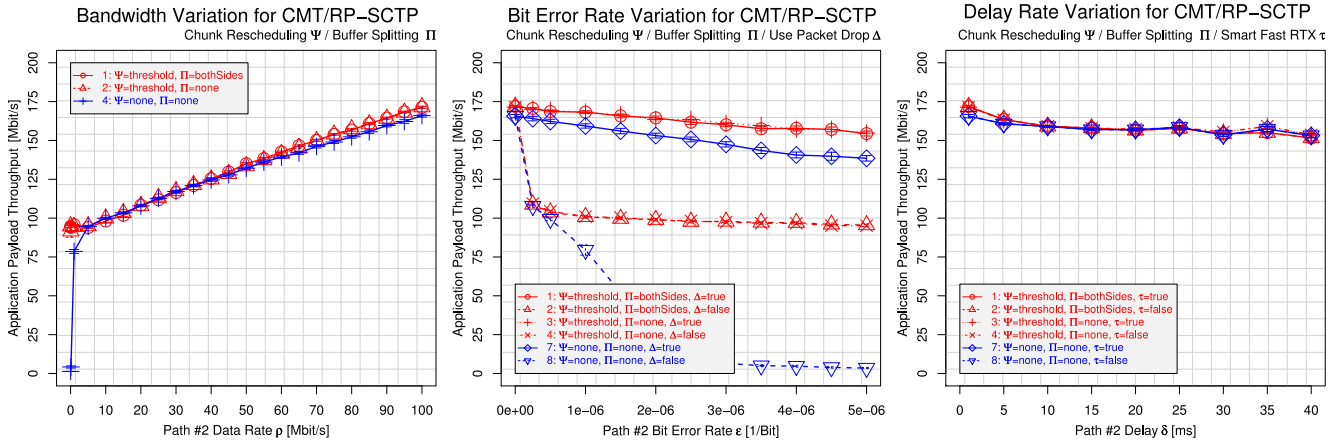

Figure 8. Ordered Transmission for CMT/RP-SCTP

VII. Conclusions

In this paper, we have examined the challenges of CMT in asymmetric setups, which are very realistic for multi-homed Internet sites. We have identified the problems of sender and receiver queue blocking, leading to poor performance of CMT. By applying our proposed mechanisms Buffer Splitting, Chunk Rescheduling and Smart Fast Retransmission, a significant performance improvement can be achieved – for plain CMT as well as for CMT combined with RP.

We are currently realizing our approaches in the FreeBSD networking stack, in order to perform experiments in lab setups as well as in a real Internet testbed. Furthermore, we are also going to contribute our results into the ongoing IETF standardization process of SCTP [14] and Multipath TCP.

References

[1] Y. Dong, D. Wang, N. Pissinou, and J. Wang, "Multi-Path Load Balancing in Transport Layer," in *Proceedings of the 3rd IEEE EuroNGI Conference on Next Generation Internet Networks*, Trondheim/Norway, May 2007, pp. 135–142, ISBN 1-4244-0857-1.

[2] J. R. Iyengar, P. D. Amer, and R. Stewart, "Concurrent Multipath Transfer Using SCTP Multihoming Over Independent End-to-End Paths," *IEEE/ACM Transactions on Networking*, vol. 14, no. 5, pp. 951–964, Oct. 2006, ISSN 1063-6692.

[3] F. Kelly and T. Voice, "Stability of End-to-End Algorithms for Joint Routing and Rate Control," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 2, pp. 5–12, Apr. 2005, ISSN 0146-4833.

[4] P. Key, L. Massoulié, and D. Towsley, "Path Selection and Multipath Congestion Control," in *Proceedings of the 26th IEEE INFOCOM*, Anchorage, Alaska/U.S.A., May 2007, pp. 143–151, ISBN 1-4244-1047-9.

[5] D. Wischik, M. Handley, and M. B. Braun, "The Resource Pooling Principle," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 5, pp. 47–52, Oct. 2008, ISSN 0146-4833.

[6] T. Dreibholz, M. Becke, J. Pulinthanath, and E. P. Rathgeb, "Applying TCP-Friendly Congestion Control to Concurrent Multipath Transfer," in *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, Perth/Australia, Apr. 2010, pp. 312–319, ISSN 1550-445X.

[7] R. Stewart, "Stream Control Transmission Protocol," IETF, Standards Track RFC 4960, Sep. 2007.

[8] P. Natarajan, N. Ekiz, E. Yilmaz, P. D. Amer, and J. Iyengar, "Non-Renegable Selective Acknowledgments (NR-SACKs) for SCTP," in *Proceedings of the 16th IEEE International Conference on Network Protocols (ICNP)*, Orlando, Florida/U.S.A., Oct. 2008, pp. 187–196, ISBN 1-4244-2507-5.

[9] R. Stewart, P. Lei, and M. Tüxen, "Stream Control Transmission Protocol (SCTP) Packet Drop Reporting," IETF, Individual Submission, Internet-Draft Version 10, Jun. 2010, draft-stewart-sctp-pktdrprep-10.txt, work in progress.

[10] J. Iyengar, P. Amer, and R. Stewart, "Receive Buffer Blocking in Concurrent Multipath Transfer," in *Proceedings of the IEEE GLOBECOM*, St. Louis, Missouri/U.S.A., Nov. 2005, pp. 121–126, ISBN 0-7803-9414-1.

[11] T. Dreibholz, M. Becke, J. Pulinthanath, and E. P. Rathgeb, "Implementation and Evaluation of Concurrent Multipath Transfer for SCTP in the INET Framework," in *Proceedings of the 3rd ACM/ICST OMNeT++ Workshop*, Málaga/Spain, Mar. 2010, ISBN 978-963-9799-87-5.

[12] T. Dreibholz, X. Zhou, and E. P. Rathgeb, "SimProcTC – The Design and Realization of a Powerful Tool-Chain for OMNeT++ Simulations," in *Proceedings of the 2nd ACM/ICST OMNeT++ Workshop*, Rome/Italy, Mar. 2009, pp. 1–8, ISBN 978-963-9799-45-5.

[13] S. Floyd and V. Jacobsen, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, 1993, ISSN 1063-6692.

[14] T. Dreibholz, M. Becke, J. Iyengar, P. Natarajan, and M. Tüxen, "Load Sharing for the Stream Control Transmission Protocol (SCTP)," IETF, Network Working Group, Internet-Draft Version 00, Jul. 2010, draft-tuexen-tsvwg-sctp-multipath-00, work in progress.